Tommi Siivola

Tietokoneohjelma säveltapailun harjoitteluun

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

14.3.2013

Metropolia

Tekijä Otsikko Sivumäärä Aika	Tommi Siivola Tietokoneohjelma säveltapailun harjoitteluun 38 sivua 14.3.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja	lehtori Olli Hämäläinen

Insinöörityön tavoitteena oli laatia tietokoneohjelma, jonka avulla voi harjoitella säveltapailua.

Insinöörityössä tutustuttiin olemassa oleviin vastaavanlaisiin ohjelmiin, ja vertailtiin niiden ominaisuuksia. Vertailua hyödynnettiin suunniteltaessa ohjelmaan toteutettavia ominaisuuksia.

Olemassa olevien ohjelmien vertailussa kävi ilmi, että ohjelmien käyttöliittymät ovat usein monimutkaisia. Tämän työn tavoitteeksi otettiinkin laatia ohjelmasta mahdollisimman helppokäyttöinen. Lisäksi tavoitteena oli soveltaa parhaita mahdollisia oppimista tukevia menetelmiä.

Ohjelman toteutustapaa valittaessa harkittiin ohjelman toteuttamista selainpohjaisena sovelluksena, Android- tai iPhone -älypuhelinsovelluksena tai Java-työpöytäsovelluksena. Toteutusalustaksi valittiin Java, sillä reaaliaikainen äänen tuottaminen oli ongelmallista muissa tutkituissa vaihtoehdoissa.

Ohjelmassa sovellettiin harjoittelutapaa, jossa harjoiteltava materiaali kuunnellaan tietyn sävellajin asiayhteydessä. Tällä menetelmällä opittuja säveltapailutaitoja voi helpommin soveltaa harjoittelutilanteen ulkopuolella.

Ohjelmassa hyödynnettiin aikavälikertausta (spaced repetition), jossa oppimisen tehostamiseksi opittavia asioita kerrataan siten, että kertauskertojen välillä on viive. Aikavälikertauksen periaatetta hyödyntämällä voidaan tehostaa laajojen aineistojen oppimista.

Ohjelman harjoitusmateriaali luotiin tietokoneavusteisesti koneluettavista musiikkiaineistoista. Oikeisiin musiikkikappaleisiin perustuvat harjoitukset toimivat harjoituksina paremmin kuin esimerkiksi satunnainen tai yksinkertaisen kaavan perusteella luotu aineisto. Koneluettavia aineistoja hyödyntämällä laadukkaan harjoitusmateriaalin luominen voitiin automatisoida.

Avainsanat	säveltapailu, Java

Author	Tommi Siivola
Title	Software application for ear training practice
Number of Pages	38 pages
Date	14 March 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Olli Hämäläinen, Senior Lecturer

The aim of this thesis was to create a software application that can be used for practicing ear training. Existing ear training applications were reviewed and their features compared as part of the work. The results of this review were taken into consideration when planning what features to implement in the application.

Review of existing programs revealed that the user interface in existing programs is often complicated. Making the application easy to use was therefore taken as an objective.

Different platforms were considered for use in implementation. The platforms that were considered were a native web application platform, Android and iPhone smartphone platforms and a Java desktop application. Java was chosen due to problems in sound generation in the other platforms.

A learning method was implemented through which the training material is listened to in the context of a musical key, which makes it easier to apply the learned ear training skills outside the training situation.

The program utilized spaced repetition in which learning is improved by inserting a time delay between repetitions of the material that is rehearsed. Spaced repetition can make the learning of large amounts of material more efficient.

Training material was generated from machine readable musical material with a computer. The use of machine readable material allowed the automatic generation of high quality training material based on real music, without much manual effort.

Keywords	ear training, Java

Sisällys

Lyhenteet

HTML *Hypertext Markup Language*. Verkkosivujen luomiseen käytetty kuvauskieli.

JSON *JavaScript Object Notation*. JavaScript-ohjelmointikielen syntaksiin perustuva yksinkertainen tiedonvälitysmuoto.

MIDI *Musical Instrument Digital Interface*. Elektronisten musiikki-instrumenttien väliseen sähköiseen tiedonvälitykseen käytettävä tiedonsiirtostandardi. MIDI-standardia tukeva elektroninen musiikki-instrumentti voi esimerkiksi lähettää tietokoneelle viestin aina kun instrumentilla soitetaan sävel.

1 Johdanto

Insinöörityön tavoitteena oli luoda helppokäyttöinen ja helposti laajennettava tietokoneohjelma, jonka avulla voi harjoitella säveltapailua.

Säveltapailu on sävelkorvan harjoittamista, jossa tavoitteena on kehittää kuulonvaraista musiikin hahmottamista, joka mahdollistaa muun muassa kuulonvaraisen soiton ja improvisoinnin. Säveltapailun tavoitteena on lopulta kehittää kyky hahmottaa pelkästään nuottien perusteella, miltä kappale kuulostaa, ja vastaavasti pelkästään kuulon perusteella hahmottaa, miten kuultu kappale kirjoitettaisiin nuoteiksi. [1, s.118; 2, s.422.]

Säveltapailuharjoittelussa perinteisesti opettaja soittaa oppilaille lyhyitä melodioita tai sointuja, ja oppilaat yrittävät tunnistaa niitä korvakuulolta. Oppilaat kirjoittavat kuulemansa nuottipaperille, jonka jälkeen heidän vastauksiaan voidaan verrata siihen, mitä opettaja soitti, ja arvioida tällä perusteella heidän taitojaan. Opettajan osuuden tästä toiminnasta voi automatisoida teknisten laitteiden avulla, jolloin oppilas ei tarvitse läsnä olevaa opettajaa voidakseen harjoitella [3].

Olemassa olevien säveltapailuharjoitusohjelmien lisäksi tämän työn innoittajina ovat olleet kielten sanastojen opetteluun käytettävät tietokoneohjelmat, kuten Supermemo ja Anki [4; 5]. Nämä ohjelmat hyödyntävät aikavälikertausta (spaced repetition), joka tarkoittaa sitä, että pelkän tavanomaisen sanojen toistamisen sijasta eri toistokertojen välillä on viive, jolloin juuri kerrattu sana kerrataan uudestaan vasta jonkin ajan kuluttua. Tämä toistojen välillä oleva viive parantaa oppimistuloksia verrattuna siihen, että sanoja toistettaisiin ilman viivettä [6]. Vaikka erilaisia tietokoneohjelmia säveltapailun harjoitteluun on olemassa, vain harvat niistä hyödyntävät aikavälikertauksen periaatetta.

Ennen työn aloittamista laadittiin suunnitelma toteutettavan ohjelman ominaisuuksista sekä työlle suuntaa antava aikataulu.

Ohjelman suunnitelluista ominaisuuksista laadittiin lista. Tarkoituksena oli toteuttaa ominaisuuksia yksi kerrallaan ja arvioida jokaisen ominaisuuden toteuttamisen jälkeen, kuinka hyvin ohjelma toimii kokonaisuutena ja mikä ominaisuus kannattaisi toteuttaa seuraavaksi.

Työn kulusta laadittiin viikoittainen aikataulu, johon kirjattiin alustavat arvioidut valmistumispäivät tärkeimmille toteutettaville ominaisuuksille. Työn edetessä edistymistä verrattiin tähän aikatauluun, jotta voitiin saada jonkinlainen käsitys siitä, miten työ etenee.

Päivittäisessä työskentelyssä myös laadittiin joka päivä suunnitelma siitä, mitä kyseisenä päivänä oli tarkoitus tehdä. Päivittäisen suunnitelman kertaaminen ja täydentäminen työskentelyn aikana auttoi keskittymään oleelliseen ja pitämään tehtävät tärkeysjärjestyksessä.

2 Katsaus olemassa oleviin ohjelmiin

Ohjelman laatimisessa otettiin huomioon olemassa olevien ohjelmien ominaisuuksia. Ohjelmista otettiin huomioon niiden hyvät puolet, jotka kannattaisi toteuttaa myös uuteen ohjelmaan. Toisaalta otettiin huomioon myös ohjelmien huonoja puolia, joita voisi välttää tai yrittää toteuttaa paremmin.

Yhteenvetona voidaan todeta, että ohjelmien yleisin ongelma on sekava käyttöliittymä. Ohjelmista löytyviä hyviä ominaisuuksia ovat harjoitusten monipuolisuus, käyttäjän ohjaaminen harjoituksesta toiseen taitojen karttuessa, sekä mahdollisuus antaa syötteet MIDI-soittimen tai jopa mikrofonin välityksellä. Hyvä ominaisuus on myös mahdollisuus itse lisätä ohjelmaan harjoituksia.

Kokonaista ohjelmaa ei tarvitse välttämättä tehdä alusta lähtien. On myös mahdollista jatkokehittää jotain olemassa olevaa ohjelmaa. Tämä kuitenkin vaatii sen, että ohjelman lähdekoodi on saatavilla ja siihen voi tehdä muutoksia. Suurimman osan tässä mainittujen ohjelmien kohdalla tämä ei kuitenkaan ole mahdollista.

Seuraavaksi käsitellään tarkemmin saatavilla olevia säveltapailun harjoitteluun tarkoitettuja ohjelmia.

2.1 GNU Solfege

GNU Solfege on virallinen osa GNU-projektia, ja se julkaistaan GNU-projektin käyttämällä GPL-lisenssillä. GNU Solfege on tehty Python-ohjelmointikielellä, ja se käyttää Gtk+ -kirjastoa graafisen käyttöliittymän luomiseen. GNU Solfegesta kokeiltiin versiota 3.20.8, joka oli tämän kirjoittamisen aikaan tuorein vakaa versio. [7.]

Ohjelma sisältää suuren määrän erilaisia harjoituksia. Harjoituksia löytyy nousevien ja laskevien intervallien, rytmien, asteikkojen sekä sointujen kuuntelemiseen. Lisäksi on teoriaharjoituksia, joissa opetellaan tunnistamaan ilmiöitä nuottikirjoituksesta.

Ohjelma sisältää myös harjoituksia, joissa opetellaan laulamaan soinnun säveliä sekä harjoituksia, joissa opetellaan kirjoittamaan kuultu melodia nuoteiksi. Näissä harjoituksissa ei kuitenkaan ole mahdollisuutta antaa ohjelman tarkistaa käyttäjän vastausta, vaan ohjelmaa voi pyytää soittamaan tai näyttämään oikean vastauksen, jonka jälkeen harjoittelijan tulee verrata omaa vastaustaan annettuun oikeaan vastaukseen ja arvioida itse, oliko vastaus oikea. Ohjelma ei myöskään ohjaa käyttäjää eteenpäin harjoituksesta toiseen, kuten oikea opettaja voisi tehdä, vaan käyttäjän täytyy itse valita lukuisista harjoituksista itselleen sopivat.

Ohjelmaan voi lisätä uusia harjoituksia ohjelman ymmärtämällä kuvauskielellä.

Ohjelman lähdekoodi on saatavilla, ja ohjelman lisenssi mahdollistaa muutoksien tekemisen.

2.2 Functional Ear Trainer

Functional Ear Trainer sisältää vain yhden harjoitustyypin [8]. Kyseessä on harjoitus. jossa säveliä kuunnellaan tiettyyn sävellajiin suhteutettuna [9; 10].

Ohjelmassa on kaksi osaa, multimediatyyppinen luento-osio, jossa selitetään ja havainnollistetaan interaktiivisen diaesityksen avulla ohjelman harjoituksen perusidea, ja harjoitteluosio, jossa voi käytännössä kokeilla harjoitusta.

Ohjelman sisältämä harjoitus vaikuttaa perusidealtaan hyvältä, mutta ohjelma ei ohjaa käyttäjää eteenpäin, kuten oikea opettaja voisi tehdä, vaan käyttäjän täytyy itse säätää harjoituksen asetukset oman osaamistasonsa mukaiseksi. Havainnollinen opastusosio on myös hyvä, koska käyttäjän ei tarvitse lukea erillisiä ohjeita, vaan ohjelman käyttö selviää ohjelmaa normaalisti käytettäessä.

Functional Ear Trainer on ilmaiseksi saatavilla, mutta ohjelman lähdekoodia ei ole saatavilla, joten ohjelmaan ei voi itse tehdä muutoksia.

2.3 Earmaster

Earmaster sisältää monia erilaisia harjoituksia, kuten intervallien, sointujen, asteikkojen, rytmien ja sointukulkujen tunnistamista. Ohjelma osaa myös vastaanottaa käyttäjän vastauksia MIDI-soittimen tai mikrofonin välityksellä. [11.]

Ohjelman käyttöliittymä sisältää monia eri asetuksia, ja tehtäviin voi antaa vastauksen monella eri tavalla. Eri asetusten ja syöttötapojen moninaisuus saattaa tuntua sekavalta.

Tehtävät on ohjelmassa järjestetty kasvavaan vaikeusjärjestykseen. Kun käyttäjä suoriutuu aiemmista tehtävistä tarpeeksi hyvin, ohjelma ehdottaa käyttäjälle uutta hieman vaikeampaa tehtävää. Ohjelman toiminta muistuttaa tältä osin oikean opettajan toimintaa.

Earmaster on saatavilla Windowsille ja Macille maksua vastaan. Ohjelmasta kokeiltiin version 6 ilmaista kokeiluversiota. Lähdekoodia ei ole saatavilla, joten ohjelmaan ei voi tehdä muutoksia.

2.4 Sibelius Auralia

Sibelius Auralia sisältää useita erilaisia harjoituksia, kuten intervallien, sointujen, sointukulkujen ja melodioiden tunnistamista. Ohjelma osaa myös vastaanottaa käyttäjän vastauksia MIDI-soittimen tai mikrofonin välityksellä. [12.]

Käyttäjä voi itse valita, mitä harjoituksia haluaa harjoitella. Tarjolla on myös mahdollisuus edetä ohjatusti asteittain haastavampiin harjoituksiin.

Ohjelma on saatavilla Windowsille ja Macille maksua vastaan. Ohjelmasta kokeiltiin ilmaiseksi saatavilla olevaa kokeiluversiota. Lähdekoodia ei ole saatavilla, joten ohjelmaan ei voi tehdä muutoksia.

3 Toteutusteknologian valinta

Ennen työn aloittamista tutkittiin vaihtoehtoisia teknologioita, joilla ohjelman voisi toteuttaa.

Ohjelma, jota käytetään jonkin asian harjoitteluun, on hyödyllinen vain siinä määrin kuin kyseistä ohjelmaa käytetään. Yksi oleellinen tekijä ohjelman hyödyllisyydessä onkin se, onko ohjelmaa ylipäätään mahdollista käyttää, esimerkiksi laitteisto- tai ohjelmistokokoonpanon rajoituksista johtuen. Ihanteellinen ratkaisu olisi monialustainen ohjelma, joka toimii kaikilla laitteilla ja sovellusalustoilla.

Yksi vaihtoehto monialustaisen ohjelman toteuttamiseksi on verkkoselainpohjainen toteutus. Selainpohjaisessa ratkaisussa ohjelmaa voi käyttää kaikkialla missä on saatavilla selainohjelma ja Internet-yhteys.

Selainpohjaisessa ratkaisussa on käytännössä kolme eri toteutusvaihtoehtoa: Javascriptin ja HTML:n yhdistelmä, Flash sekä Java-sovelma.

Javascript ja HTML on ihanteellisin vaihtoehto, koska silloin käyttäjän ei tarvitse erikseen asentaa ylimääräisiä lisäosia omalle laitteelleen, vaan kaikki tarvittava on selaimessa valmiina. Vaikuttaa kuitenkin siltä, että nykyinen teknologia ei ole riittävän kehittynyttä, jotta lähes reaaliaikainen äänen tuottaminen olisi mahdollista Javascriptin ja HTML:n avulla. Javascriptin ja HTML:n yhdistelmän äänentuotto-ominaisuuksia selvitettiin kokeilemalla SoundManager 2 Javascript -kirjaston ominaisuuksia esittelevää rumpukoneohjelmaa sekä Chromium-selaimen ääniominaisuuksia esitteleviä ohjelmia [5; 6]. Ongelmana näissä oli, että kyseiset esimerkkiohjelmat joko eivät toimineet kaikilla selaimilla tai sitten äänentoisto oli katkonaista ja nykivää, tai ohjelma reagoi käyttäjän toimintaan liian hitaasti.

Flash-sovellus tai Java-sovelma eivät kärsi samassa määrin Javascript-sovelluksen ongelmista reaaliaikaisen äänen tuottamisen kanssa, mutta haittapuolena on se, että ne vaativat lisäosien asentamista käyttäjän laitteelle.

Toinen ongelma, joka liittyy kaikkiin selainpohjaisiin sovelluksiin, on tiedon tallentaminen. Ohjelman on tarkoitus kerätä tietoa käyttäjän toiminnasta ja säädellä toimintaansa tämän tiedon perusteella. Tiedon tallentaminen on tärkeää, jotta ohjelma voi esimerkiksi muistaa, mitä harjoituksia käyttäjä on tehnyt edellisenä päivänä ja mukauttaa toimintaansa sen perusteella. Selainpohjaisissa sovelluksissa tiedon tallentamiseen vaaditaan pääsääntöisesti palvelinpuolen ohjelmisto, joka tallentaa tiedon ja on yhteydessä käyttäjän selaimessa toimivaan asiakaspuolen ohjelmaan. Erillisten palvelin- ja asiakasohjelmien toteuttaminen lisää ohjelman monimutkaisuutta. Ongelmana on myös, että ohjelmaa käyttääkseen palvelimen täytyy olla toiminnassa ja käyttäjällä täytyy olla yhteys palvelimeen. Toimintavarmuuden kannalta paras ratkaisu onkin kokonaan käyttäjän laitteella toimiva ratkaisu, joka ei ole riippuvainen verkkoyhteyden tai ulkoisen palvelimen toiminnasta.

Toinen hyvä vaihtoehto toteutusalustaksi on jokin älypuhelinalusta, kuten Android tai iPhone. Älypuhelimet ovat nykyään suosittuja, ja älypuhelimessa toimivan sovelluksen etuna on, että se voi olla käyttäjän mukana missä tahansa.

Android sisältää syntetisaattorin, joka voi soittaa sille lähetettyjä MIDI-käskyjä, mutta tätä kirjoitettaessa Androidin ohjelmointirajapinnat eivät tarjoa suoraan pääsyä tähän syntetisaattoriin. Sovelluskehittäjille on tarjolla vain rajoitetumpi rajapinta, joka ei mahdollista reaaliaikaista äänentuottoa. [13.]

Applen iPhone- ja iPad-alustoilla reaaliaikainen äänentuotto toimisi todennäköisesti paremmin, mutta tähän työhön ei ollut käytettävissä Applen laitteistoa kehitystä ja testausta varten, joten tämä vaihtoehto ei tullut kyseeseen.

Lopulta toteutusalustaksi valittiin selainpohjaisen tai älypuhelinalustan sijasta Java. Java-sovellukset on helppo saada toimimaan eri alustoilla, jos alustalle on saatavissa Java-virtuaalikone. Java-virtuaalikoneessa on myös sisäänrakennettuna syntetisaattori, jota voi käyttää reaaliaikaiseen äänentuottoon. Lisäksi Javalle on saatavilla JFugue-kirjasto, joka helpottaa musiikin soittamista Javan syntetisaattorilla [14]. Itsenäinen Java-sovellus on myös ylläpidon kannalta helpompi kuin esimerkiksi verkkosovellus, sillä itse sovellus ja sen tallettamat tiedot voivat olla käyttäjän koneella, joten ohjelman käyttö ei vaadi yhteyttä palvelimeen.

4 Harjoitusten suunnittelu

Itse ohjelman ydin on varsinainen säveltapailun harjoittelu, jota käyttäjä ohjelman avulla tekee. Ohjelmaan suunniteltiin kolme erityyppistä harjoitusta, jotka harjoittavat kuulon eri osa-alueita.

Perusperiaate säveltapailun harjoittelussa tietokoneohjelman avulla on, että tietokoneohjelma soittaa säveliä käyttäjälle, joka kuuntelee sävelet ja kertoo käyttöliittymän kautta ohjelmalle mitä kuuli. Tämän jälkeen ohjelma tarkistaa, vastasivatko käyttäjän ilmoittamat sävelet sitä, mitä oikeasti soitettiin eli kuuliko käyttäjä sävelet oikein. Tämän tiedon perusteella ohjelma tekee joitain toimenpiteitä, kuten antaa käyttäjälle palautetta ja valitsee mitkä sävelet käyttäjälle seuraavaksi soitetaan.

Ohjelman soittamat sävelet voivat periaatteessa olla mitä tahansa säveliä tai sävelryhmiä, kuten sointuja, murrettuja sointuja tai intervalleja, riippuen täysin siitä, mitä sävelkorvan osa-aluetta halutaan kehittää. Esimerkiksi yksittäisten sävelten tunnistaminen vaatii erilaista harjaantumista kuin useista sävelistä koostuvien sointujen tunnistaminen.

Tässä käytetään sanaa ”tehtävä” toiminnosta, jossa ohjelma soittaa käyttäjälle säveliä ja käyttäjä syöttää ohjelmalle oman vastauksensa. Sanalla ”harjoitus” tarkoitetaan useasta tehtävästä koostuvaa kokonaisuutta. Harjoitus on siis jatkuvaa toimintaa, jossa käyttäjän tehtäväksi annetaan uusia tehtäviä yksi toisensa jälkeen.

Yksittäinen harjoitus keskittyy tiettyyn kuuntelemisen osa-alueeseen ja koostuu useasta tähän osa-alueeseen liittyvästä tehtävästä. Harjoitus voi esimerkiksi keskittyä duurisävellajin sävelkulkujen tunnistamiseen, jolloin jokainen tehtävä sisältää yhden duurisävellajissa yleisesti esiintyvän sävelkulun, jonka käyttäjän täytyy tunnistaa. Vastaavasti harjoitus voi keskittyä mollisävellajin sointujen ja sointukulkujen tunnistamiseen, jolloin jokainen tehtävä sisältää soinnun tai useasta peräkkäisestä soinnusta muodostuvan sointukulun, jonka käyttäjän tulee tunnistaa.

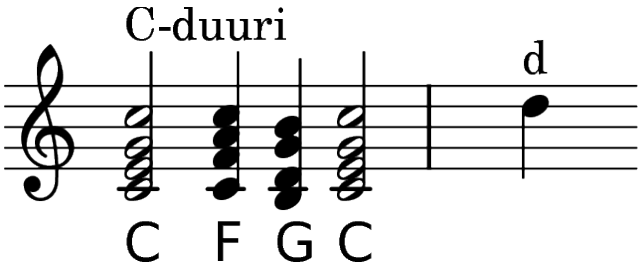
Kun ohjelma havaitsee, että käyttäjä osaa vastata tiettyyn tehtävään oikein, kyseinen tehtävä esitetään käyttäjälle harvemmin ja tilalle otetaan uusia tehtäviä, joita käyttäjä ei vielä hallitse yhtä hyvin. Näin käyttäjän aika käytetään mahdollisimman tehokkaasti, painottaen sellaista aineistoa, joka vaatii enemmän harjoitusta.

Harjoitusten laatimisen lähtökohdaksi otettiin harjoitus, joka on kuvattu Paula Telescon Journal of Music Theory Pedagogy -lehdessä julkaistussa artikkelissa sekä muusikko Frank Singerin artikkelissa. Kyseisessä harjoituksessa harjoittelijalle soitetaan ensin sointukulku, joka virittää kuulijan tiettyyn sävellajiin, ja välittömästi tämän sointukulun soittamisen jälkeen ne sävelet, jotka harjoittelijan tulee tunnistaa. [9; 10.]

Musiikki on erittäin voimakkaasti asiayhteydestä riippuva ilmiö. Jokaisella sävelellä on oma värähtelytaajuutensa, mutta se, miltä sävel kuulostaa riippuu lopulta siitä, minkä muiden sävelten kanssa se esiintyy yhdessä, eli siitä mitä muita säveliä soi säveltä ennen, sen jälkeen ja sen kanssa samanaikaisesti.

Sointukululla voidaan saada aikaan vaikutelma sävellajista, ja tällaisen sointukulun soittaminen ennen varsinaisten tunnistettavien sävelten soittamista asettaa tunnistettavat sävelet tämän sävellajin asiayhteyteen. Tällainen sävellajiin suhteutettu kuuntelemisen harjaannuttaminen on hyödyllistä, koska käytännössä musiikkia kuunnellessa sävelet kuullaan aina suhteessa johonkin sävellajiin. Tätä ajatusta sovellettiin harjoitusten suunnittelussa siten, että jokaisessa harjoituksessa tehtävän aluksi soitetaan sointukulku, joka virittää kuulijan tiettyyn sävellajiin, ja varsinaiset sävelet, jotka tulee tunnistaa soitetaan tämän virittäytymisen jälkeen.

Kuvassa 1 on nuottiesimerkki, jossa sointukulku saa aikaan vaikutelman C-duurisävellajista, jonka jälkeen soitetaan yksittäinen d-sävel. Tämä d-sävel kuullaan suhteutettuna C-duuri sävellajiin, koska se kuullaan välittömästi C-duurisävellajin vaikutelman aikaansaavan sointukulun jälkeen.



Kuva 1. Esimerkki harjoituksesta jossa aluksi sointukulku virittää kuulijan tiettyyn sävellajiin

Lisäksi on tärkeää, että tehtävät ovat musikaalisia. Esimerkiksi yksittäisten sävelten tai sointujen tunnistamista harjoitellessa on hyödyllistä, jos nämä sävelet tai soinnut esiintyvät jonkinlaisessa musikaalisessa yhteydessä, koska tällöin harjoiteltu taito siirtyy paremmin harjoitusten ulkopuoliseen maailmaan [9, s. 179–183]. Sävellajin virittävien sointukulkujen lisäksi tätä periaatetta sovellettiin siten, että tehtävien tunnistettavista sävelistä pyrittiin luomaan musikaalisesti järkeviä kokonaisuuksia.

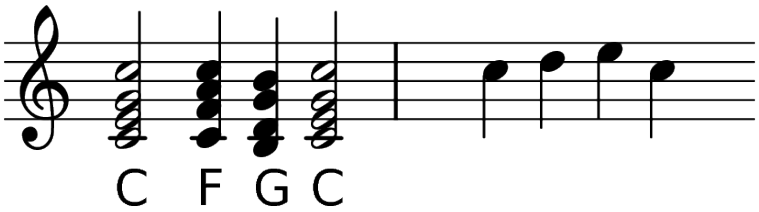
Ohjelmaan laadittiin kolme erilaista harjoitusta: harjoitus peräkkäisillä sävelillä, harjoitus samanaikaisilla sävelillä ja harjoitus soinnuilla.

Seuraavaksi esitellään tarkemmin eri harjoitukset.

4.1 Harjoitus peräkkäisillä sävelillä

Harjoituksessa peräkkäisillä sävelillä harjoitellaan yksi toisensa jälkeen peräkkäin soitettujen sävelten eli melodioiden tunnistamista. Oikeat musiikissa esiintyvät melodiat voivat koostua jopa kymmenistä peräkkäisistä sävelistä. Tässä harjoituksessa lähestytään melodioiden kuuntelua yksinkertaistettujen muutamasta sävelestä koostuvien melodiakatkelmien kautta. Yksinkertaisimmillaan harjoituksen melodiakatkelmassa voi olla vain yksi sävel. Sävelten määrä kasvaa harjoittelun edetessä, kun käyttäjän hahmotuskyky kehittyy.

Harjoituksen kulku on sellainen, että ohjelma soittaa ensin sointukulun ja sitten yhden tai useamman sävelen peräkkäin, jonka jälkeen käyttäjän tulee käyttöliittymän avulla ilmoittaa ohjelmalle, mitkä sävelet olivat kyseessä. Kuvassa 2 on nuottiesimerkki tämän harjoituksen tehtävästä, jossa käyttäjälle aluksi soitetaan sointukulku ja tämän jälkeen neljästä peräkkäisestä sävelestä koostuva melodiakatkelma.



Kuva 2. Esimerkki tehtävästä jossa tunnistetaan peräkkäisiä säveliä.

Harjoituksen tehtävien laatimisessa päätettiin aluksi käyttää hyödyksi Paula Telescon artikkelissa olevaa taulukkoa yleisimmin musiikissa esiintyvien sävelten siirtymistä, joita Telesco hyödyntää omassa opetuksessaan. Taulukossa on listattu, miltä sävelasteelta useimmin siirrytään millekin toiselle sävelasteelle. Tehtävät voisivat periaatteessa koostua täysin satunnaisesti valituista sävelistä, mutta musiikissa yleisesti esiintyvien ilmiöiden kokoaminen yhteen ja hyödyntäminen opiskelussa voi lisätä opiskelun tehokkuutta, kun opittavat asiat ovat jollakin tavalla kytköksissä siihen maailmaan, jossa opeteltavia taitoja on tulevaisuudessa tarkoitus käyttää. [9, s.183–185.]

Telescon listaamat sävelsiirtymät päätettiin aluksi sisällyttää tehtäviin sellaisenaan. Ongelmaksi muodostui, että Telescon listassa on vain sävelpareja, mutta näiden harjoitusten kokeilemisen yhteydessä kävi selväksi, että mielenkiintoinen ja monipuolinen harjoitus vaatii, että tehtävissä on enemmän kuin vain kaksi säveltä peräkkäin. Tätä ongelmaa yritettiin ratkaista laatimalla Telescon listan perusteella tehtäviä, joissa on kahden sävelen sijasta kolme säveltä, periaatteena se, että Telescon listaa voi ajatella eräänlaisena sääntökokoelmana, joka määrää jokaisen sävelen kohdalla mikä sävel voi seurata sitä. Tällaisen sääntökokoelman avulla voi luoda periaatteessa rajattoman pitkiä ketjuja, joissa sävelet seuraavat toisiaan noudattaen näitä sääntöjä. Kuitenkin kokeiltaessa ohjelmaa näillä kolmen sävelen tehtävillä todettiin, että ne eivät tuntuneet mielenkiintoisilta ja luonnollisilta, vaan pikemminkin mekaanisilta.

Kolmen sävelen tehtävien mekaanisuus johtui oletettavasti siitä, että ne eivät tarpeeksi realistisesti noudattaneet oikean musiikin sääntöjä. Telescon listan käyttäminen sääntökokoelmana, jonka perusteella voi luoda melodioita, oli todennäköisesti liian yksinkertainen lähestymistapa.

Seuraava ajatus oli analysoida oikean musiikin sävelkulkuja ja luoda tehtävät näiden pohjalta. Tähän tarkoitukseen löydettiin Jack Campinin ylläpitämä sivusto, jonne hän oli koonnut ABC-tiedostomuotoon tallentamiaan vanhoja kansansävelmiä [15].

ABC-tiedostomuoto on yksinkertainen tekstipohjainen esitystapa, jolla voidaan kuvata melodioita. Se on kehitetty apuvälineeksi musiikista keskustelemiseen Internetissä tekstipohjaisten sovellusten, kuten sähköpostin, välityksellä. ABC-tiedostomuotoa käytetään yleisesti kansanmusiikkikappaleiden välittämiseen erilaisilla verkkosivustoilla ja harrastajien postituslistoilla. [16]

Koodiesimerkissä 1 on esimerkki siitä, miltä ABC-muotoinen tiedosto näyttää. ABC-tiedosto on tekstitiedosto, jossa on ilmaistu kappaleen nimi ja muita tietoja sekä kappaleen nuotit tekstipohjaisella merkitsemistavalla. Koodiesimerkissä 1 ensimmäiset rivit ovat kappaleen metatietoja, viimeiset neljä riviä puolestaan sisältävät varsinaisen musiikin. Esimerkin viimeisillä neljällä rivillä jokainen kirjain vastaa yhtä nuottia ja numerot ilmaisevat nuottien pituuksia. Pystyviivat ovat tahtiviivoja ja kaksoispisteet kertausmerkkejä. Nämä kaksi jälkimmäistä merkintätapaa muistuttavat ulkonäöltään perinteisessä nuottikirjoituksessa olevia merkintätapoja. Tahtiviiva on perinteisessä nuottikirjoituksessa myös pystyssä oleva viiva ja kertausmerkki kaksi päällekkäin olevaa pistettä.

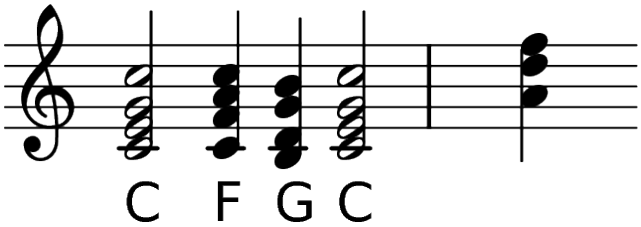
```
X:0001
T:The Ranting Highlandman.
T:The White Cockade
M:C|
L:1/8
Q:1/2=112
I: :: ::
%% G A B c d e ^f g a
Z:Jack Campin * www.campin.me.uk * 2009
K:G
AG|B2B2 B2AG|B2B2 B2g2|B2B2 B2AG|AGAB A2
GA|B2B2 cBAG|A2B2 g2fg|a2gf g2fe|d2B2 B2::
Bc|d2B2 g2B2|d2d2 d2e2|d2cB g2fg|a2A2 A2
GA|B2B2 cBAG|A2B2 g2fg|a2gf g2fe|d2B2 B2:|
```

Koodiesimerkki 1. Esimerkki ABC-muotoisesta musiikkikappaleesta. Kappale on Jack Campinin Aird's Airs kokoelmasta [15].

Campinin sivustolta valittiin "Aird's Airs" -niminen kokoelma, johon kuuluu 1180 James Airdin vuosina 1778–1801 julkaisemaa kappaletta [15]. Harjoitukset laadittiin yksinkertaisen ohjelman avulla, joka luki nämä sävelmät, poimi niistä sävelkulkuja ja loi näistä sävelkuluista tehtäviä säveltapailuohjelmaan.

Näitä tehtäviä kokeiltaessa todettiin niiden olevan huomattavasti parempia kuin aiemmin laaditut Telescon listaan perustuvat tehtävät.

4.2 Harjoitus samanaikaisilla sävelillä



Kuva 3. Esimerkki tehtävästä jossa tunnistetaan yksittäisiä säveliä äänimassasta.

Harjoitus samanaikaisilla sävelillä on muuten samanlainen kuin harjoitus peräkkäisillä sävelillä, mutta tässä harjoituksessa alun sointukulun jälkeen soitetut sävelet soitetaan kaikki yhtä aikaa, jolloin ne muodostavat soinnun tai äänimassan, josta käyttäjän täytyy opetella kuulemaan yksittäiset sävelet. Samanaikaisesti soivien sävelten kuuntelu ja toisistaan erottaminen toimii vastapainona yksitellen soitettujen sävelten kuuntelulle. Kuvassa 3 on esimerkki yhdestä tähän harjoitukseen kuuluvasta tehtävästä. Aluksi käyttäjälle soitetaan sointukulku, joka luo vaikutelman C-duurisävellajista, jonka jälkeen käyttäjälle soitetaan useammasta sävelestä koostuva äänimassa, tässä esimerkissä sävelistä a, d ja f koostuva sointu.

Kahteen tämän tyyppiseen harjoitukseen luotiin tehtävät, yhdet tehtävät duurisävellajiin ja toiset mollisävellajiin. Tehtävät luotiin käsin, valitsemalla säveliä, jotka muodostavat sävellajin sointuja niiden eri käännöksinä.

4.3 Harjoitus soinnuilla

Harjoituksessa soinnuilla alun sointukulun jälkeen soitetaan sointuja. Harjoitus on muuten samanlainen kuin harjoitus peräkkäisillä sävelillä, mutta yksittäisten sävelten tilalla on sointuja, jotka sävelkulkujen sijaan muodostavat sointukulkuja.

Koska musiikin koneellinen analyysi toimi hyvin laadittaessa tehtäviä peräkkäisten sävelten harjoitukseen, päätettiin kokeilla samaa lähestymistapaa myös sointuharjoituksen tehtävien luomisessa. Tähän tarkoitukseen löydettiin Trevor De Clercqin ja David Temperleyn rock-musiikkia käsittelevän tutkimusprojektin verkkosivusto [17]. Osa kyseistä tutkimusprojektia oli ollut suosituimpien rock-kappaleiden harmonioiden analysointi ja kirjaaminen koneluettavaan muotoon myöhempää tilastollista käsittelyä varten. Tämä koneluettava sointuanalyysimateriaali oli saatavilla sivustolla, joten sitä käytettiin hyväksi sointutehtävien laadinnassa.

Samaan tapaan kuin peräkkäisten sävelten harjoituksen tehtävien laatimisessa, myös tämän harjoituksen tehtävät laadittiin yksinkertaisen ohjelman avulla. Ohjelma luki sointuanalyysit, erotteli niistä sointukulkuja ja laati näistä sointukuluista tehtävät harjoitukseen.

5 Käyttäjän edistymisen seuranta

Jotta harjoittelu olisi mahdollisimman tehokasta, ohjelma pitää kirjaa käyttäjän oikeista ja vääristä vastauksista ja tarjoaa käyttäjälle useammin tehtäviä, joihin käyttäjä on aiemmin vastannut väärin, sillä oletuksella, että nämä ovat niitä tehtäviä, joissa käyttäjä tarvitsee lisää harjoitusta. Samoin ohjelma siirtää taka-alalle tehtävät, joihin käyttäjä on aiemmin vastannut useasti oikein, ja antaa näitä käyttäjän tehtäväksi harvemmin. [6.]

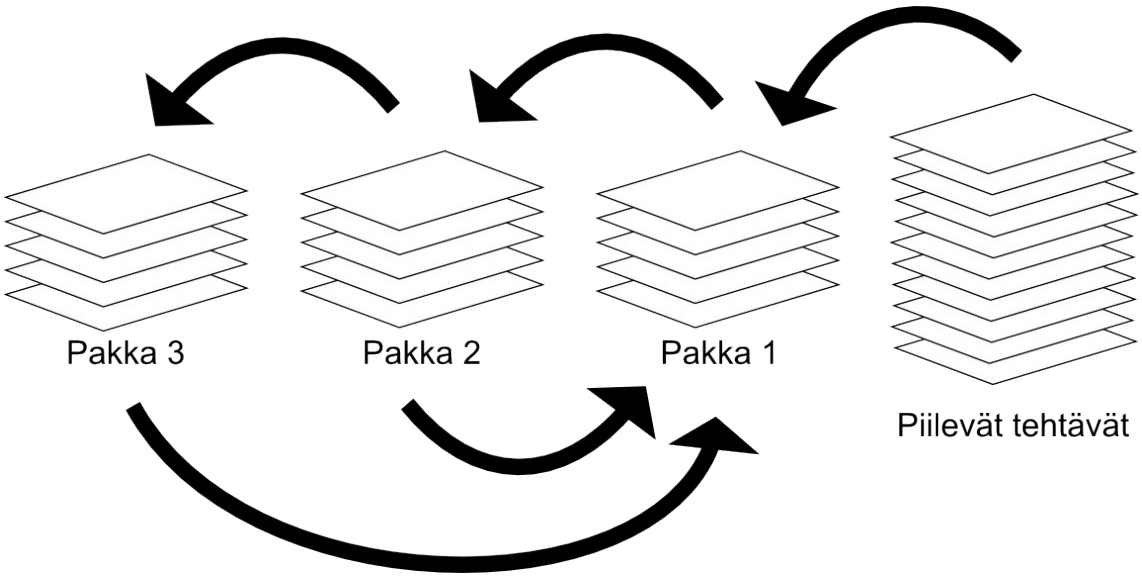
Menetelmä, jolla ohjelma valitsee tehtäviä, perustuu Sebastian Leitnerin kehittämään sanakorttimenetelmään (flash-card). Kyseistä menetelmää käytetään usein esimerkiksi kielten opiskelun yhteydessä sanastojen opetteluun. Menetelmässä opeteltavat sanat kirjoitetaan korteille, ja näistä korteista muodostetaan pakkoja. Eri pakoissa olevia sanoja kerrataan eri aikavälein. Vaikeiden sanojen pakassa olevia kortteja kerrataan useammin, kun taas helpompien sanojen pakassa olevia sanoja kerrataan harvemmin. Kun jokin vaikeiden sanojen pakassa oleva kortti tulee tutuksi ja muuttuu helpommaksi, se siirretään eri pakkaan. Näin harjoittelu tulee tehokkaasti keskitettyä niihin sanoihin, jotka vaativat eniten harjoittelua, mutta samalla helpommat sanat eivät pääse unohtumaan, koska niihinkin palataan ajoittain, joskin harvemmin. [18.]

Ohjelmassa jokaisen harjoituksen tehtävät on jaettu eri kategorioihin, jotka vastaavat sanakorttimenetelmän pakkoja. Ohjelma antaa tehtäviä käyttäjän tehtäväksi ja pitää kirjaa kunkin tehtävän kohdalla käyttäjän antamista oikeista ja vääristä vastauksista. Jos oikeiden vastausten määrä jonkin tehtävän kohdalla ylittää tietyn rajan, kyseinen tehtävä siirretään helpompien tehtävien kategoriaan. Vastaavasti jos käyttäjä vastaa tehtävään useasti väärin, kyseinen tehtävä siirretään takaisin vaikeampien tehtävien kategoriaan. Näin harjoitus mukautuu käyttäjän edistymiseen tai taantumiseen eri tehtävissä.

Kun käyttäjä aloittaa harjoituksen ensimmäisen kerran, ei vielä tiedetä, mitkä tehtävät ovat käyttäjälle helppoja tai vaikeita. Kaikki tehtävät ovatkin aluksi erillisessä piilevien harjoitusten kategoriassa, josta tehtäviä ei suoraan valita käyttäjän tehtäväksi. Piilevien harjoitusten kategoria sisältää ikään kuin koko opinto-ohjelman, ja sieltä otetaan uusia

asioita harjoiteltavaksi aina, kun ohjelma toteaa käyttäjän vastausten perusteella, että tämä hallitsee aiemmin harjoitellut asiat tarpeeksi hyvin. Kun käyttäjä suoriutuu hänelle annetuista tehtävistä menestyksellä, ohjelma siirtää tehtäviä helpompiin kategorioihin, ja vaikeimpien tehtävien kategoria tyhjenee. Kun vaikeimpien tehtävien kategoriassa olevien tehtävien määrä laskee alle tietyn rajan, otetaan sinne lisää harjoituksia piilevien harjoitusten kategoriasta.

Kuvassa 4 on havainnollistettu tehtävien siirtymistä eri kategorioiden välillä. Kuvassa on kolme kategoriaa eli pakkaa, joista valitaan tehtäviä käyttäjän tehtäväksi. Lisäksi on piilevien tehtävien pakka. Tehtävät ovat aluksi piilevien tehtävien pakassa, mistä ne siirtyvät pakkaan 1. Kun käyttäjä vastaa tehtävään oikein, se siirtyy seuraavaan pakkaan, kunnes lopulta päätyy pakkaan 3. Jos taas käyttäjä vastaa tehtävään väärin, kyseinen tehtävä siirtyy takaisin pakkaan 1. Pakassa 1 olevia tehtäviä annetaan käyttäjän tehtäväksi useimmin, pakassa 2 olevia hieman harvemmin ja pakassa 3 kaikista harvimmin. Helpot tehtävät, joihin käyttäjä osaa vastata oikein ja jotka täten vaativat vähemmän harjoittelua, päätyvät pakkaan 3, kun taas vaikeat tehtävät, jotka vaativat enemmän harjoittelua, päätyvät pakkaan 1.

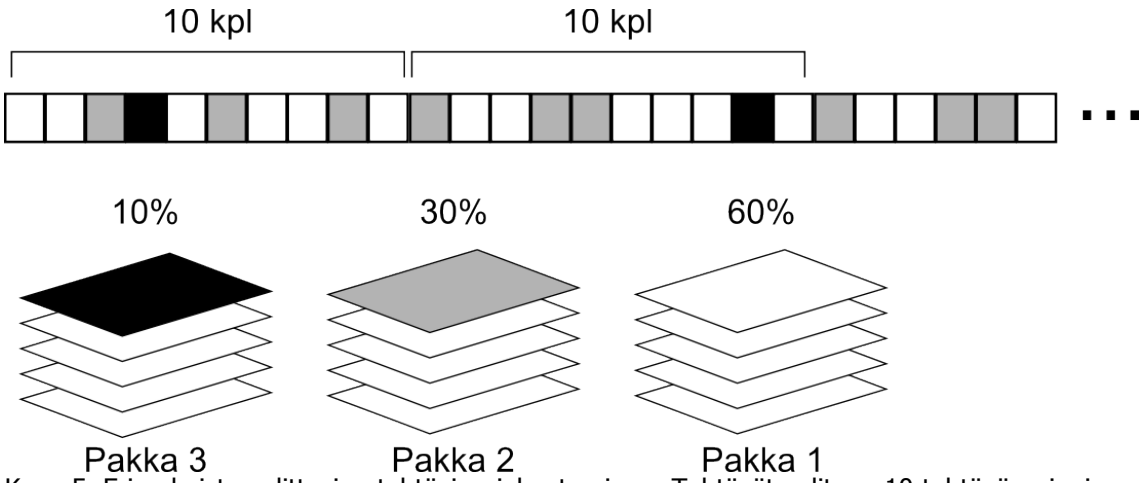


Kuva 4. Tehtävien siirtyminen pakasta toiseen.

Artikkelissa The Application of Cognitive Scientific Findings Related to Spacing Effects to Web-Based CAI Systems esitellään kolme periaatetta, joiden avulla aikavälikertaukseen perustuvaa harjoittelua voidaan tehostaa [19].

Ensimmäisen periaatteen mukaan tehokkaan harjoittelun aikaansaamiseksi on käyttäjälle tarjottavien tehtävien jakauduttava siten, että vaikeita tehtäviä annetaan käyttäjän tehtäväksi useammin kuin helppoja tehtäviä. Toinen periaate sanoo, että tehtävää, johon on vastattu oikein muutaman kerran peräkkäin, ei enää kannata kerrata, sillä oppimistulokset eivät parane useista peräkkäisistä toistoista, jos harjoittelijalla on jo kyseisen tehtävän oikea vastaus lyhytkestoisessa muistissa. Kolmas periaate sanoo, että jos vääriä vastauksia tulee useita peräkkäin, tulee nämä väärin vastatut tehtävät kerrata nopeasti uudelleen. Usea peräkkäinen väärä vastaus voi olla merkki siitä, että tehtävien välinen viive on liian pitkä, jolloin opitut asiat ehtivät haihtua harjoittelijan lyhytkestoisesta muistista.

Ensimmäinen periaate on toteutettu ohjelmassa siten, että käyttäjän tehtäväksi valitaan pieni tehtävänippu, joka koostuu eri kategorioista valituista tehtävistä, siten että enemmän harjoitusta vaativien tehtävien kategorioista valikoituu enemmän tehtäviä kuin vähemmän harjoitusta vaativien kategorioista. Kun käyttäjä on käynyt kaikki tehtävänipun tehtävät läpi, valitaan samalla periaatteella uusi nippu. Yksi mahdollinen esimerkki tehtävien valitsemisesta voisi olla sellainen, jossa tehtävät on jaettu kolmeen eri kategoriaan, ja nippuun valitaan kerrallaan tehtäväksi kymmenen tehtävää. Kuvassa 5 on havainnollistettu tämä esimerkki kuvaamalla eri pakoista valittuja tehtäviä eri väreillä. Nämä kymmenen tehtävää valitaan niin, että helpoimpien tehtävien kategoriasta valitaan yksi tehtävä, seuraavaksi helpoimpien kategoriasta kolme tehtävää ja vaikeimpien kategoriasta kuusi tehtävää. Kun nämä kymmenen tehtävää on valittu, ne annetaan käyttäjän tehtäväksi. Kun käyttäjä on tehnyt nämä kymmenen, valitaan uudet kymmenen samalla periaatteella. Näin varmistetaan, että käyttäjälle tarjotut tehtävät jakautuvat siten, että 60 % tulee vaikeimpien, 30 % keskivaikeiden ja 10 % helpoimpien tehtävien kategoriasta, jolloin keskimäärin vaikeimpien kategorioiden tehtäviä tulee käyttäjälle vastaan useammin. Käyttäjä tekee tehtäviä kuitenkin yksi kerrallaan peräkkäin, joten käyttäjä ei välttämättä ole tietoinen siitä, että ohjelma valitsee tehtäviä tietyn kokoisissa ryhmissä.



Kuva 5. Eri pakoista valittavien tehtävien jakautuminen. Tehtävät valitaan 10 tehtävän nipuissa, joihin valitaan tehtäviä eri pakoista tietyssä suhteessa.

Toinen periaate on toteutettu ohjelmassa siten, että jos sama tehtävä esiintyy tehtävänipussa useasti ja käyttäjä vastaa tähän tehtävään useasti oikein, niin tämän tehtävän muut esiintymät poistetaan nipusta. Tällöin kyseinen tehtävä ei tule käyttäjälle vastaan kuin vasta sen jälkeen kun uusi tehtävänippu on koostettu.

Kolmas periaate on toteutettu ohjelmassa siten, että jos käyttäjä vastaa tehtäviin väärin useasti peräkkäin, keskeytetään nykyisen tehtävänipun suorittaminen ja luodaan uusi tehtävänippu. Koska tehtävät, joihin on vastattu väärin, esiintyvät tehtävänipussa useammin kuin muut, esiintyvät väärin vastatut tehtävät myös tässä uudessa tehtävänipussa useammin, jolloin nämä tehtävät tulevat kerratuksi useammin.

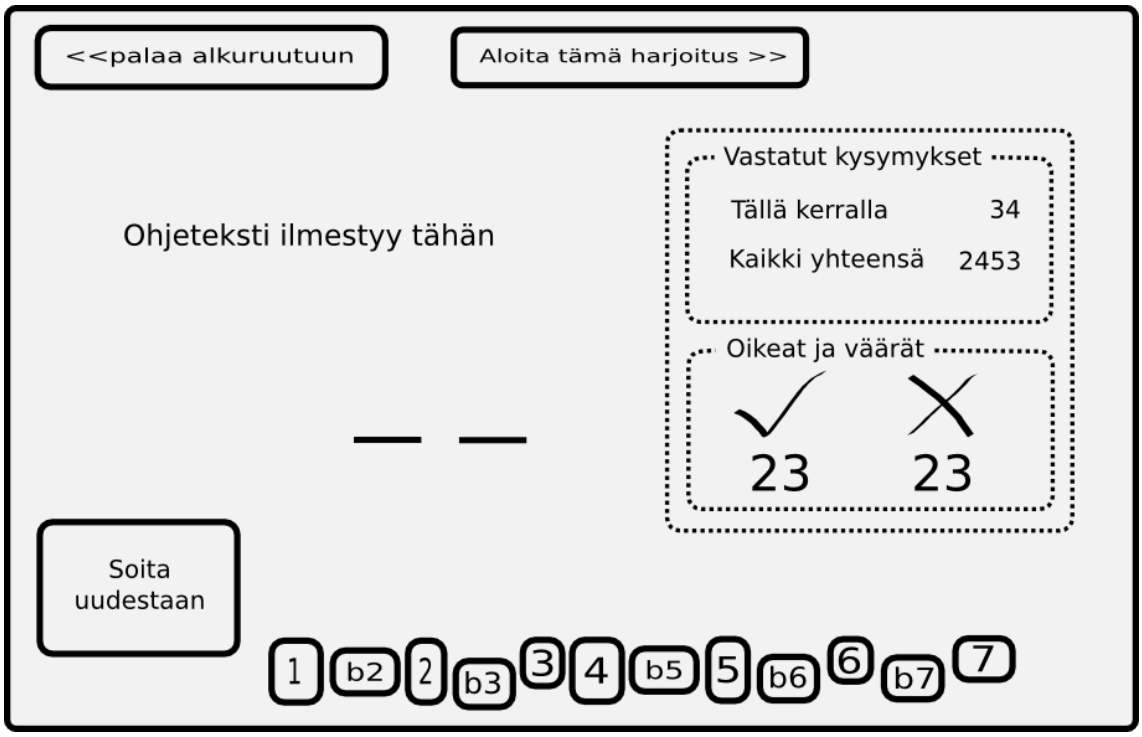
6 Käyttöliittymän suunnittelu

Käyttöliittymä on tämänkaltaisessa sovelluksessa tärkeä, sillä ohjelman käyttö perustuu lähes pelkästään käyttöliittymään, toisin kuin joissain toisissa sovellustyypeissä, joissa käyttöliittymää käytetään ehkä vain ajoittain joidenkin asetusten tekemiseen, jonka jälkeen ohjelma toimii itsenäisesti.

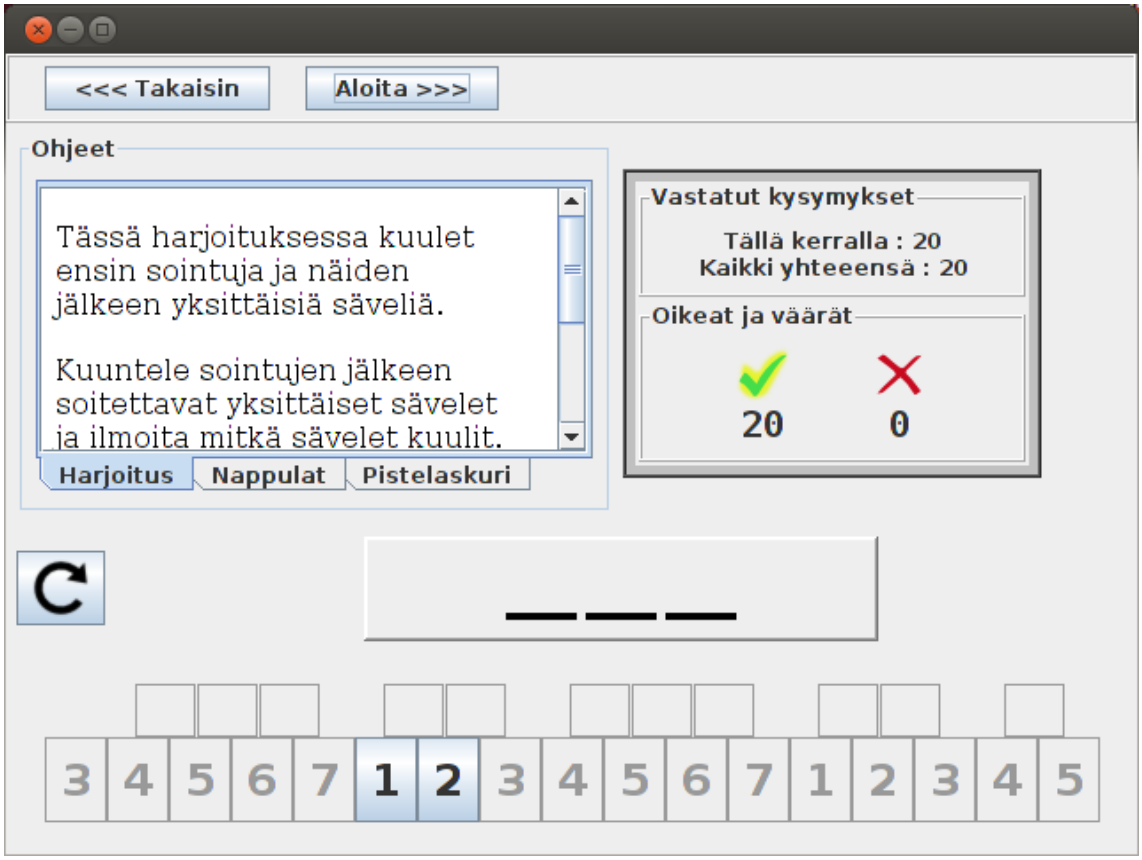
Käyttöliittymän suunnittelussa noudatettiin työtapaa, jossa ensin piirrettiin käyttöliittymästä kuvia, joiden perusteella arvioitiin käytettävyyttä ja selkeyttä. Toteuttaminen aloitettiin vasta, kun piirtämällä oli saatu hahmoteltua soveltuva käyttöliittymä. Tällä toteutustavalla pyrittiin siihen, että toteutuksen tekniset yksityiskohdat eivät päässeet liiaksi vaikuttamaan käyttöliittymän ulkoasuun, vaan käyttöliittymä voitiin suunnitella vapaasti piirtämällä ilman teknisiä rajoituksia.

Piirtämiseen käytettiin InkScape-vektorigrafiikkaohjelmaa [20]. Myös muutamat ikonit, joita ohjelma käyttää, on piirretty InkScapella.

Kuvassa 6 on esimerkki siitä, miltä suunnitteluvaiheen piirros käyttöliittymästä näytti. Kuvassa 7 puolestaan on ruutukaappaus valmiin ohjelman käyttöliittymästä.



Kuva 6. Piirros käyttöliittymästä.



Kuva 7. Ruutukaappaus valmiista ohjelmasta.

Käyttöliittymän suunnittelun periaatteena oli, että käyttäjä voi keskittyä varsinaiseen tehtävään eli säveltapailun harjoitteluun, eikä hänen tarvitse tehdä mitään muuta, kuten säätää asetuksia tai lukea erillisiä ohjeita. Harjoittelun voi aloittaa kahdella napsautuksella, ja tarvittavat ohjeet ovat näkyvissä kun ohjelmaa käytetään.

Ohjelma koostuu ikkunasta, jossa näytetään eri näkymiä. Jokaiselle harjoitustyypille on oma näkymä, ja lisäksi on valikkonäkymä, josta voi valita harjoituksen.

Eri harjoitustyyppien näkymät ovat suurelta osin samanlaisia. Kaikissa harjoitusnäkymissä on ohjelaatikko, josta voi lukea ohjeita, sekä pistelaatikko, josta käyttäjä näkee viimeisimmät oikeat ja väärät vastaukset. Tärkein ero eri harjoitustyyppien näkymien välillä on osio, jolla käyttäjä syöttää vastauksensa ohjelmalle. Koska joissain harjoituksissa käyttäjä kuuntelee ja tunnistaa säveliä yksitellen ja toisissa taas yksittäisiä sointuja kokonaisuuksina, tarvitaan eri väline sävelten syöttämiseen ja sointujen syöttämiseen.

Seuraavaksi käsitellään käyttöliittymän eri osa-alueita tarkemmin.

6.1 Ohjelaatikko

Ohjelaatikko on esillä näkyvällä paikalla, jotta käyttäjä voi helposti lukea ohjeita, silloin kun tuntee niitä tarvitsevansa. Ohjelaatikkoon pitää mahtua kaikki tarvittavat ohjeet, mutta itse laatikko ei saa viedä suhteettoman suurta osaa käyttöliittymän tilasta. Tästä syystä ohjelaatikossa on välilehdet, jotta samasta laatikosta voi lukea ohjeita eri aiheista, ja vierityspalkit siltä varalta, että tietty aihe ei mahdu kokonaan laatikkoon.

Alun perin suunnitelmana oli, että ohjeet toimisivat älykkäästi mukautuen kulloiseenkin tilanteeseen. Tarkoituksena oli, että ohjelma olisi neuvonut käyttäjää jokaisen uuden tilanteen yhteydessä tilanteeseen sopivalla tavalla. Esimerkiksi käyttäjän aloittaessa ohjelman käytön ensimmäistä kertaa, käyttäjälle olisi automaattisesti näytetty ohjeet, joissa olisi neuvottu, mitä painikkeita tulee painaa päästäkseen alkuun. Kokeiluvaiheessa kuitenkin havaittiin, että älykkäästi mukautuvat ohjeet eivät tarjonneet merkittävää lisähyötyä, sillä ohjelman käyttö oli jo valmiiksi melko helppoa, joten älykkäiden ohjeiden toteuttamisesta luovuttiin.

6.2 Pistelaatikko

Pistelaatikko antaa käyttäjälle tietoa harjoituksen edistymisestä. Pistelaatikko antaa välittömän palautteen siitä, oliko viimeisimpään tehtävään annettu vastaus oikein vai väärin. Lisäksi pistelaatikko näyttää nykyisen harjoituskerran oikeiden ja väärien vastausten lukumäärän. Pistelaatikko näyttää myös, kuinka monta tehtävää on käyty läpi sekä nykyisellä harjoituskerralla että kaikkina menneinä harjoituskertoina yhteensä.

Käyttäjä voi käyttää tietoa meneillään olevalla harjoituskerralla suoritettujen tehtävien määrästä harjoituskerran pituuden säätämiseen. Käyttäjä voi esimerkiksi asettaa tavoitteekseen suorittaa tietty määrä tehtäviä harjoituskerralla ja pistelaatikon tietojen avulla seurata tavoitteen täyttymistä.

Kaikkina menneinä harjoituskertoina läpikäytyjen tehtävien kokonaismäärä antaa käyttäjälle kuvan edistymisestä pidemmällä aikavälillä.

6.3 Sävelvalitsin

Ohjelmassa tarvitaan käyttöliittymän osio, jonka avulla käyttäjä voi ilmoittaa kuulemiaan säveliä ohjelmalle. Tästä käyttöliittymän osiosta käytetään tässä nimeä sävelvalitsin. Sävelvalitsimessa täytyy olla oma painike jokaiselle sävelelle, joita painelemalla käyttäjä voi syöttää vastauksensa.

Yksi tärkeä periaate käyttöliittymän suunnittelussa oli, että käyttäjä saa vastaustaan syöttäessä välittömästi musikaalisen palautteen siitä, mitä hän on syöttänyt. Sävelvalitsimen kohdalla tämä tarkoittaa sitä, että ohjelma soittaa välittömästi sen sävelen, jota vastaavaa painiketta käyttäjä painaa. Näin käyttöliittymä on ikään kuin musikaalinen instrumentti, jolla käyttäjä soittaa vastauksensa. Samalla vastauksen syöttämisestä tulee osa harjoittelua, kun käyttäjä kuuntelee, vastaavatko hänen syöttämänsä vastauksen sävelet ohjelman aiemmin soittamaa tehtävää.

Eri säveliä on 12, kun lasketaan kaikki kromaattisen asteikon sävelet, joten sävelvalitsimessa täytyy olla vähintään 12 eri painiketta. Käytännössä painikkeita täytyy kuitenkin olla enemmän, sillä 12 säveltä sisältyy vain kromaattisen asteikon yhteen oktaaviin, ja kunnolliset harjoitukset vaativat laajemman kuin vain yhden oktaavin äänialan hyödyntämistä.

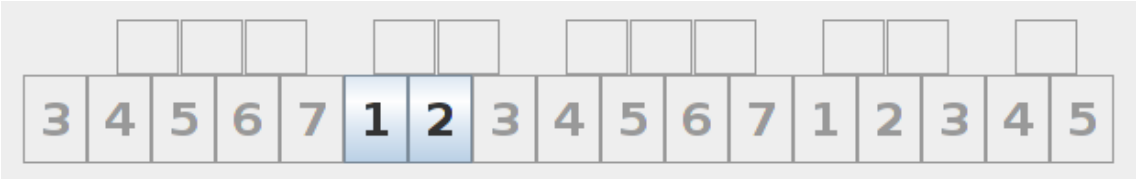
Suunnitteluvaiheessa harkittiin muutamia erilaisia tapoja sävelvalitsimen toteuttamiseksi.

Ensimmäisessä suunnitelmassa painikkeet oli sijoiteltu peräkkäin riviin ja eri oktaavien painikkeet olivat eri riveillä. Tällaisen järjestelyn etu on, että monta painiketta saadaan sovitettua tiiviiseen tilaan, sillä uusien oktaavien lisääminen ei kasvata painikeryhmän kokoa leveyssuunnassa. Toinen etu on, että eri oktaavit ovat havainnollisesti esitettynä.

Toisessa suunnitelmassa kunkin painikkeen suhde sävellajiin oli merkitty painikkeiden erilaisella koolla, sijainnilla ja värillä. Kaikki sävelet, jotka kuuluvat sävellajin perussävelestä alkavaan duuriasteikkoon, olisivat olleet edustettuina samanvärisillä tai samalle korkeudelle sijoitetuilla painikkeilla, ja vastaavasti molliasteikkoon kuuluvien sävelten painikkeilla olisi oma värinsä tai sijaintinsa. Ajatuksena oli, että tällainen hahmottamisen lakeja noudattava jaottelu voisi helpottaa sävelten hahmottamista käyttöliittymässä [21, s.117–120].

Vaihtoehtona oli myös, että lopullinen sävelvalitsin muodostuisi kahden edellisen suunnitelman yhdistelmästä, jossa painikkeet olisivat eri riveillä oktaavien perusteella ja jokaisen rivin painikkeet olisivat ryhmitelty värien tai sijaintien perusteella. Tällaista järjestelyä arvioitaessa todettiin kuitenkin, että eriväriset ja -kokoiset painikkeet vaikuttivat sekavilta. Eri riveillä sijaitsevien painikkeiden haittapuolena puolestaan oli, että rivin viimeinen painike ja seuraavan rivin ensimmäinen painike olivat kaukana toisistaan, vaikka käsitteellisesti ne ovat asteikolla vierekkäisiä. Harjoituksissa nämä sävelet esiintyivät usein peräkkäin, mutta käyttöliittymässä säveliä vastaavat painikkeet olivat kaukana toisistaan, mikä teki vastaamisesta hieman hankalampaa.

Lopulta painikkeet päädyttiin järjestämään perinteistä pianon koskettimistoa muistuttavaan järjestykseen, jossa niin sanotut "valkoisten koskettimien" sävelet ovat rivissä, ja niin sanotut "mustien koskettimien" sävelet näiden yläpuolella lomittain. Tällainen järjestely, jossa painikkeet olivat johdonmukaisesti rivissä, oli vähemmän sekavan näköinen, ja toisiaan käsitteellisesti lähellä olevat sävelet olivat painikerivissä myös edustettuina lähekkäin. Lisäksi pianon koskettimia muistuttava asettelu on todennäköisesti jollakin tapaa tuttu suurelle osalle ihmisiä, joten heillä on ennestään jonkinlainen käsitys siitä, miten sitä on tarkoitus käyttää. Kuvassa 8 on kuvakaappaus sävelvalitsimesta valmiissa ohjelmassa. Kuvassa näkyy, että vain kaksi painiketta, numeroidut 1 ja 2, ovat aktiivisina. Säätämällä mitkä painikkeet ovat aktiivisia voidaan rajata käyttäjän vastausvaihtoehtoja, ja täten rajata ja ohjata käyttäjän huomiota, ja helpottaa tehtäviä. Käyttäjän edistyessä yhä useammat painikkeet muuttuvat aktiivisiksi, ja vastaavasti tehtävät muuttuvat haastavammaksi, sillä käyttäjän täytyy huomioida enemmän asioita.



Kuva 8. Pianon koskettimia muistuttava sävelvalitsin. Kuvakaappaus valmiista ohjelmasta.

6.4 Sointuvalitsin

Sointuharjoituksia varten ohjelma tarvitsee käyttöliittymään sävelvalitsinta vastaavan osion, jolla käyttäjä voi syöttää ohjelmalle sointuja. Tästä käyttöliittymän osiosta käytetään tässä nimitystä sointuvalitsin.

Kuten sävelvalitsimen tapauksessa, myös sointuvalitsin antaa välittömän musikaalisen palautteen käyttäjän syöttäessä vastaustaan.

Sointuharjoituksissa soinnut voidaan tunnistaa kahden perusominaisuuden, soinnun pohjasävelen sekä soinnun tyypin perusteella. Tästä johtuen sointuvalitsimen toteuttamisvaiheessa ilmeni toteutukselle kaksi luonnollista vaihtoehtoa. Yksi vaihtoehto oli, että soinnun pohjasävelen ja tyypin valitseminen olisi erotettu toisistaan siten, että

pohjasävelen valintaan on omat painikkeensa ja soinnun tyypin valintaan myöskin omansa, jolloin yksittäisen soinnun valitseminen vaatii kahden painikkeen painallusta. Toinen vaihtoehto oli, että jokaiselle soinnun pohjasävelen ja tyypin yhdistelmälle olisi oma painikkeensa. Jälkimmäisen vaihtoehdon huono puoli on siinä, että painikkeita tulee paljon. Jokaista harjoituksessa käytettyä sointutyyppiä kohden täytyy olla 12 painiketta, jotta saadaan katettua jokainen eri pohjasävel kyseiselle sointutyypille, jolloin jokaisen sointutyypin lisääminen harjoitukseen kasvattaisi tarvittavien painikkeiden määrää 12:lla. Ensimmäisessä vaihtoehdossa puolestaan sointutyyppien lisääminen kasvattaisi painikkeiden lukumäärää vain yhdellä jokaista lisättyä sointutyyppiä kohden. Jälkimmäinen ratkaisu valittiin toteutuksen lähtökohdaksi, koska se vaatii vain yhden painikkeen painamista sointua valitessa ja on täten helppokäyttöisempi.

Painikkeet järjestettiin ruudukkomuodostelmaan siten, että eri sointutyyppien painikkeet ovat omilla riveillään. Pohjasävelet ovat järjestettynä siten, että vierekkäisten painikkeiden pohjasävelet ovat toisistaan kvintin päässä. Tällainen järjestely on käytössä haitarin bassopuolen koskettimissa. Sen etuna on, että musiikissa yleisesti lähekkäin esiintyviä sointuja vastaavat painikkeet ovat myöskin lähellä toisiaan. Kuvassa 9 on kuvakaappaus sointuvalitsimesta valmiissa ohjelmassa.

Kuva 9. Sointuvalitsin. Kuvakaappaus valmiista ohjelmasta.

6.5 Moniosaisen vastauksen näyttö

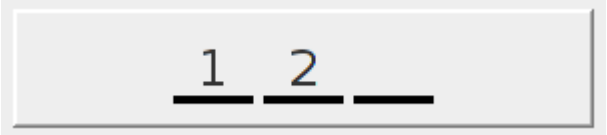
Harjoituksien luonteeseen kuuluu, että yksittäinen tehtävä voi koostua useasta kohteesta, jotka käyttäjän täytyy yksitellen syöttää ohjelmalle. Kun tehtävä koostuu useasta kohteesta, täytyy käyttäjän myös syöttää vastauksensa useassa osassa. Tällaisessa tilanteessa käyttäjälle on hyvä antaa jatkuvasti välitöntä palautetta, jotta hän voi hahmottaa, mitä hän on jo syöttänyt, ja mitä häneltä vielä odotetaan [21, s.134]. Tätä pa-

lautteen antamista varten käyttöliittymässä on oma osionsa, jossa näkyy, kuinka monta kohdetta käyttäjän odotetaan tunnistavan ja mitkä kohteet käyttäjä on jo syöttänyt ohjelmalle.

Kuvissa 10, 11 ja 12 näkyvät moniosaisen vastauksen näytön eri vaiheita, kun käyttäjä syöttää vastaustaan. Käyttäjän vastauksen eri osat tulevat näkyviin viivoille. Viivat myös kertovat käyttäjälle, kuinka montaa eri osaa vastauksessa on. Kuvassa 10 näkyy tilanne, jossa käyttäjä ei vielä ole syöttänyt mitään. Kuvassa 11 käyttäjä on antanut ensimmäisen syötteen, joka näkyy ensimmäisellä viivalla. Kuvassa 12 näkyy tilanne käyttäjän toisen syötteen jälkeen.

Kuva 10. Moniosaisen vastauksen näyttö ennen käyttäjän syötettä.

Kuva 11. Moniosaisen vastauksen näyttö käyttäjän ensimmäisen syötteen jälkeen.



Kuva 12. Moniosaisen vastauksen näyttö käyttäjän toisen syötteen jälkeen.

7 Tekninen toteutus

Ohjelman toteutettiin Java-ohjelmointikielellä. Käyttöliittymä on toteutettu Javaan sisältyvällä Swing-ikkunointikirjastolla sekä kolmannen osapuolen MigLayout-kirjastolla [22]. Äänien soittamiseen ohjelma käyttää Javan virtuaalikoneeseen sisältyvää MIDI-syntetisaattoria ja JFugue-kirjastoa [14]. Ohjelman käyttämät asetustiedostot ovat JSON-muodossa, joiden lukemiseen ohjelma käyttää Jackson-kirjastoa [23].

7.1 Äänien soittaminen Javan syntetisaattorilla

Ensimmäinen asia, jota alettiin toteuttaa, oli äänien soittaminen, sillä tämä on ohjelman kaikkein välttämättömin osa. Javan virtuaalikoneen syntetisaattorin käyttö sellaisenaan on melko monimutkaista, joten apuna päätettiin käyttää Jfugue-kirjastoa, joka tekee syntetisaattorin käytöstä yksinkertaisempaa.

Toteutuksen ensimmäisessä vaiheessa kokeiltiin Javan syntetisaattorin ja Jfugue-kirjaston toimintaa. Kokeiluja tehdessä havaittiin, että Jfuguen julkaisuversiossa 4.0.3 oli ongelmia. Ääni toistui välillä katkonaisesti, ja useampaa säveltä peräkkäin toistettaessa kaikki sävelet kuuluivatkin yhtä aikaa. Kehitysversio 4.1:llä ääni toistui ilman katkoja ja sävelet soivat oikeissa paikoissa, mutta tässä versiossa ilmeni toinen ongelma, nimittäin ääni toistui viiveellä, ja viive tuntui kasvavan joka kerta, kun ohjelma soitti jonkin äänijakson. Kokeiluohjelmassa oli painike, jota painamalla ohjelma soitti muutamasta sävelestä koostuvan jakson. Ensimmäisellä kerralla painiketta painettaessa sävelet lähtivät soimaan välittömästi, mutta jokaisella seuraavalla painalluskerralla sävelet lähtivät soimaan vasta viiveen jälkeen, ja viive kasvoi joka painalluksella pidemmäksi. Selvitystyön aikana paljastui, että Jfuguen sisäinen ajankulua seuraava laskuri alkoi ajautua väärään aikaan, jos Jfuguella soitti useammasta sävelestä koostuvan äänijakson. Soitto alkoi myöhässä, koska ajankulusta vastaava laskuri näytti väärää aikaa ja ajasti siksi soiton alkamisen väärin.

Ongelma saatiin korjattua muokkaamalla Jfuguen 4.1 version lähdekoodia siten, että soittoa aloitettaessa sävelten ajastuksesta vastaavat laskurit asetetaan oikeaan aikaan.

Lopullisessa ohjelmassa äänien soittaminen on koteloitu omaan luokkaansa. Tämä luokka huolehtii syntetisaattorin alustamisesta sekä soitettavien sävelten välittämisestä Jfugue-kirjastolle.

7.2 Ohjelman arkkitehtuuri

Ohjelma jakautuu MVC-mallin mukaisesti kolmeen osaan. MVC-mallissa ohjelman toimintalogiikka ja käyttöliittymä on jaettu erillisiin osiin, joita kutsutaan malliksi (Model) ja näkymäksi (View), jotka yhdistää toisiinsa kolmas osa, käsittelijä (Controller). MVC-mallin etuna on, että ohjelman toimintalogiikka ja käyttöliittymä voidaan toteuttaa erillisinä. Tällöin esimerkiksi toimintalogiikasta vastaavan mallin rajapintojen muutokset eivät vaadi muutoksia käyttöliittymästä vastaavaan näkymään, sillä näkymä ei ole suoraan yhteydessä mallin rajapintoihin, vaan yhteys toimii välissä olevan käsittelijän välityksellä. [24, s.142–144.]

Ohjelmassa jokainen harjoitus muodostaa oman MVC-mallin mukaisen kokonaisuuden. Jokaisella harjoituksella on käsittelijänä toimiva olio, joka luo sekä näkymä-olion että malli-olion ja yhdistää nämä toisiinsa. Ohjelman päänäkymään laitetaan esille kulloinkin aktiivisen harjoituksen näkymä. Lisäksi on olemassa näkymä, joka kuvaa päävalikkoa. Päänäkymässä komponenttien sijoittelusta huolehtii Swing-kirjastosta löytyvä CardLayout-luokan ilmentymä. CardLayout-luokan avulla yhteen komponenttiin voidaan sijoittaa useita alikomponentteja, joista yksi voidaan valita näkyväksi kerrallaan, hieman kuin korttipakassa vain päällimmäinen kortti on kerrallaan näkyvillä [25].

Harjoituksen malli-komponentti vastaa kysymysten soittamisesta käyttäjälle, käyttäjän vastausten tarkistamisesta sekä käyttäjän edistymisen seurannasta ja tallentamisesta levylle. Näkymä-komponentin kautta käyttäjä puolestaan syöttää vastauksensa ohjelmalle ja saa ohjelmalta palautetta. Näkymä-komponentti ja malli-komponentti toteuttavat tarkkailija-suunnittelumallin [24, s.142–144]. Näkymän eri osiot, jotka reagoivat käyttäjän toimintaan, kuten painikkeet, voivat rekisteröidä kuuntelijaolioita, jotka suorittavat toimintoja, kun esimerkiksi käyttäjä painaa painiketta. Samoin malli-komponentti voi rekisteröidä kuuntelijaolioita, jotka suorittavat toimintoja, kun mallin tilassa tapahtuu muutoksia. Harjoituksen käsittelijä-komponentti kytkee mallin ja näkymän toisiinsa luomalla ja rekisteröimällä kummallekin sopivat kuuntelijaoliot siten, että kun

käyttäjä painelee painikkeita näkymässä, niin mallin tiettyjä metodeita kutsutaan. Samoin kun mallin olotilassa tapahtuu muutoksia, esimerkiksi malli toteaa, että käyttäjän viimeisin vastaus oli oikea, niin mallin rekisteröimä kuuntelija kutsuu näkymän metodeita sillä tuloksella, että käyttäjälle välittyy palaute oikeasta vastauksesta.

7.3 Tehtävien sisäinen esitys ja käsittely

Jokaisen harjoitustyypin yksittäinen tehtävä koostuu joukosta kohteita, jotka esitetään käyttäjälle ja jotka käyttäjän täytyy sen jälkeen tunnistaa ja ilmoittaa käyttöliittymän välityksellä ohjelmalle. Sävelkulku- ja sävelmassaharjoituksissa nämä tunnistettavat kohteet ovat yksittäisiä säveliä, kun taas sointukulkuharjoituksissa ne ovat yksittäisiä sointuja.

Koska tehtävät muistuttavat perusluonteeltaan toisiaan, päätettiin, että tehtävät esitetään ohjelman sisäisessä toiminnassa numeroina ja numeroista koostuvina listoina. Sävelkulku- ja sävelmassaharjoituksissa jokaisella sävelellä on oma numeronsa, ja puolestaan sointukulkuharjoituksissa jokaisella soinnulla on oma numeronsa. Tämä ratkaisu mahdollistaa sen, että tehtävien käsittely voidaan hoitaa yleisillä numeroita käsittelevillä mekanismeilla, joita voidaan uudelleenkäyttää eri harjoitustyypeissä.

Kun harjoitus käynnistyy, harjoituksen malli-komponentti lataa numerolistoista koostuvat tehtävät harjoituksen asetustiedostosta muistiin. Kun käyttäjä syöttää vastaustaan käyttöliittymän painikkeilla, syöttää jokainen painikkeen painallus taustalla olevalle malli-komponentille numeroita, joista kootaan numerolista. Näitä numeroita vertaamalla selvitetään, onko käyttäjän vastaus oikea vai väärä. Samat mekanismit ovat toiminnassa harjoituksen tyypistä riippumatta.

7.4 Tilanteen tallennus levylle

Ohjelma tallentaa harjoituksen tilanteen levylle, jotta harjoittelua voi jatkaa samasta pisteestä myöhemmin.

Harjoitukset on toteutettu siten, että jokaisessa harjoituksessa kaikki kyseiseen harjoitukseen liittyvä tallennusta vaativa tieto sijaitsee yhdessä oliossa, erillään varsinaisesta harjoituksen toimintaan liittyvästä logiikasta. Lisäksi ohjelmassa on oma tiedon tallentamisesta vastaava luokka, joka pystyy sarjallistamaan olion, ja tallentamaan sarjallistetun tiedon levylle. Kun tulee aika tallentaa harjoituksen tilanne levylle, tämä harjoituksen tiedon sisältävä olio annetaan tallentamisesta vastaavalle oliolle, joka tallentaa tiedon levylle. Vastaavasti kun ohjelma käynnistetään, jokainen harjoitus pyytää tiedon tallentamisesta vastaavaa oliota noutamaan kyseiseen harjoitukseen liittyvän aiemmin tallennetun tiedon levyltä.

Kun sarjallistettu tieto luetaan levyltä, ja siitä luodaan olio ohjelman muistiin, täytyy ohjelmakoodista löytyvän luokkamääritelmän olla yhteensopiva sarjallistetun tiedon kanssa. Tallennettavan tiedon erottaminen omaan olioonsa siten, että vain tämän olion sisältämä tieto tallennetaan, rajaa tämän yhteensopivuusvaatimuksen koskemaan vain tämän olion luokkaa. Tämä mahdollistaa muun ohjelmakoodin vapaamman muuntelun ilman vaaraa, että aiheutuu ongelmia yhteensopivuudessa tallennetun tiedon kanssa.

8 Uusien harjoitusten lisääminen ohjelmaan

Yksi tavoite ohjelman toteutuksessa oli, että ohjelmaa olisi myöhemmin helppo laajentaa uusilla harjoituksilla. Laajennettavuutta varten ohjelma suunniteltiin siten, että eri harjoitusten tiedot luetaan erillisistä asetustiedostoista. Tällöin uusien harjoitusten lisääminen ei vaadi ohjelmakoodin muuttamista, vaan riittää, että luodaan ja muokataan asetustiedostoja.

Ohjelman asetustiedostot käyttävät JSON-muotoa, joka on JavaScript-ohjelmointikielen syntaksiin perustuva tiedonsiirtomuoto [23]. JSON-muotoisten tiedostojen lukemiseen ohjelma käyttää Jackson-kirjastoa [26].

Yleinen käytäntö Java-sovelluksissa on käyttää asetustiedostoissa XML-muotoa, mutta tässä päädyttiin käyttämään JSON-muotoa, koska se on yksinkertaisempi, joten sitä on helpompi lukea ja kirjoittaa käsin. Lisäksi JSON-muotoista tulostetta on helppo tuottaa yleisillä skriptauskielillä, kuten Pythonilla ja Rubylla, sillä näiden kielten perustietorakenteiden syntaksi muistuttaa JSON:in tietorakenteiden syntaksia.

Jokainen harjoitus on kuvattu omassa JSON-tiedostossa. Tiedostossa on kuvattu harjoituksen tietoja, kuten nimi. Lisäksi tiedosto sisältää kaikki harjoitukseen kuuluvat tehtävät. Myös ohjelman alkunäkymän harjoitusvalikon rakenne on määritetty omassa JSON-muotoisessa asetustiedostossa. Uuden harjoituksen voi lisätä ohjelmaan luomalla uuden JSON-tiedoston, jossa harjoitus on kuvattu, ja muokkaamalla alkunäkymän harjoitusvalikon asetustiedostoa siten, että uusi harjoitus tulee näkyviin valikkoon.

```
{
"identifier":"major_conseq",
"description" :"Duuri",
"cadences": "major",
"instructions": [
["Harjoitus", "tonesequence_instructions.html"],
["Nappulat", "solfege_buttons.html"],
["Pistelaskuri", "score_instructions.html"]
],
"questions":
[[12],
 [14],
 [12, 14],
 [14, 12],
 [12, 14, 12],
 [14, 12, 14],
 [14, 12, 14, 12],
 [12, 14, 12, 14]
]}
```

Koodiesimerkki 2. Esimerkki harjoituksen määrittelevästä asetustiedostosta.

Koodiesimerkissä 2 on esimerkki siitä, miltä harjoituksen JSON-muotoinen asetustiedosto näyttää. JSON-muotoon kuuluu kaksi eri tietorakennetta, taulukko (array) ja olio (object), ja näiden lisäksi merkkijono- ja numero-tietotyypit. Numerot esitetään sellaisenaan, merkkijonot lainausmerkkien sisällä. Taulukko-tyyppinen tietorakenne esitetään hakasulkeina (**[ ]**), joiden sisällä on pilkulla eroteltuna taulukon alkiot. Olio-tyyppinen tietorakenne esitetään aaltosulkeina (**{ }**), joiden sisällä on pilkulla eroteltuna avain-arvo-pareja. Avain-arvo-parin avain ja arvo on erotettu toisistaan kaksoispisteellä (**:**). Taulukko- ja olio-tietotyypit ovat rekursiivisia, eli esimerkiksi taulukko voi sisältää toisia taulukoita tai olioita, jotka voivat edelleen sisältää toisia taulukoita tai olioita, tai myös muun tyyppisiä alkioita. [23.]

Koodiesimerkissä 2 nähdään, mitä tietoja harjoitukseen voi kuulua. Kenttä nimeltä ”identifier” on harjoituksen yksilöllinen tunnus. Tämän tunnuksen perusteella harjoitus erotetaan muista harjoituksista, ja tämä tunnus määrää esimerkiksi, minkä nimiseen tiedostoon tämän harjoituksen tiedot tallennetaan. Kenttä ”description” on harjoituksen kuvaus, joka näkyy käyttöliittymässä. Kenttä ”cadences” määrittää, mitä sointukulkuja jokaisen tehtävän alussa soitetaan. Kenttä ”instructions” määrittää, mitä ohjetiedostoja harjoituksen ohjelaatikossa näytetään. Kenttä ”questions” sisältää harjoituksen tehtävät. Jokainen tehtävä on taulukollinen numeroita. Harjoituksen tyypistä riippuen nämä

numerot merkitsevät joko peräkkäin soitettavia säveliä, soinnun tai äänimassan säveliä tai peräkkäin soitettavia sointuja.

Koodiesimerkissä 3 on asetustiedosto joka määrittää ohjelman alkuvalikon. Asetustiedosto koostuu useista JSON-olioista. Jokainen olio määrittää yhden kohdan alkuvalikossa. Oliot, joiden ”item”-kenttä sisältää arvon ”excercise”, luovat alkuvalikkoon painikkeen, josta voi siirtyä kyseiseen harjoitukseen. Puolestaan oliot, joiden ”item”-kenttä sisältää arvon ”section”, luovat alkuvalikkoon osion, joka sisältää painikkeita. Kaikki tällaista ”item”-oliota seuraavat painikemääritykset näkyvät valikossa kyseisen osion sisällä.

```
[
{"item":"section", "label":"Melodiaharjoitukset"},
{"item":"excercise", "type":"solfege",  "settings_file":"ma-
jor.json"},
{"item":"excercise", "type":"solfege",  "settings_file":"mi-
nor.json"},

{"item":"section", "label":"Äänimassaharjoitukset"},
{"item":"excercise", "type":"tonemass",
"settings_file":"major_key_chords.json"},
{"item":"excercise", "type":"tonemass", "settings_file":"ma-
jor_key_intervals.json"},

{"item":"section", "label":"Harmoniaharjoitukset"},
{"item":"excercise", "type":"harmony", "settings_file":"ma-
jor_key.json"},
{"item":"excercise", "type":"harmony", "settings_file":"mi-
nor_key.json"},
]
```

Koodiesimerkki 3. Esimerkki päävalikon asetustiedostosta.

9 Yhteenveto

Tässä insinöörityössä laadittiin tietokoneohjelma, jonka avulla voi itsenäisesti harjoitella säveltapailua.

Toteutusalustaksi harkittiin selainpohjaista ratkaisua tai älypuhelinalustaa, mutta näitä kokeiltaessa ilmeni ongelmia reaaliaikaisen äänentuoton kanssa. Ohjelma päätettiin toteuttaa Java-työpöytäsovelluksena, koska Java-sovelluksen saa toimimaan monella eri alustalla, ja reaaliaikainen äänentuotto on mahdollista. Työpöytäsovellus ei kuitenkaan ole nykyaikana kaikkein ihanteellisin alusta tämäntyyppiselle sovellukselle, joka vaatii käyttäjältä pitkäaikaista aktiivista käyttöä. Älypuhelinsovellus olisi ihanteellisin vaihtoehto, mutta ainakin Android-alustan puutteet alustaan sisältyvän MIDI-syntetisaattorin ohjelmointirajapinnoissa asettavat toteutukselle rajoituksia. Iphone- ja Windows Phone -alustojen mahdollisuuksia ei tässä työssä tutkittu.

Tämän työn aikana keskityttiin perustoiminnallisuuden luomiseen. Esimerkiksi sovelluksen ulkoasun viimeistelyyn ei käytetty kovin paljoa aikaa. Sovelluksen ulkoasua ja käyttöliittymää voisikin kehittää edelleen.

Ohjelma pitää kirjaa käyttäjän vastauksista ja valitsee käyttäjälle uusia tehtäviä melko yksinkertaisella korttipakkamenetelmällä. Tällä menetelmällä voidaan yksinkertaisesti toteuttaa aikavälikertauksen periaate, mutta tämä menetelmä on alun perin kehitetty käytettäväksi ilman tietokonetta, joten se on kompromissi yksinkertaisuuden ja oppimistulosten tehokkuuden välillä. Kun käytössä on tietokone, ei menetelmän monimutkaisuus aiheuta ongelmia, sillä tietokone pystyy suoriutumaan monimutkaisista ja työläistäkin toimenpiteistä nopeasti ja virheettömästi. Yksinkertaisen korttipakkamenetelmän korvaaminen hienostuneemmalla tehtävienvalinta-algoritmilla voisikin parantaa ohjelmalla saavutettavia oppimistuloksia.

Ohjelma antaa käyttäjälle palautetta siitä, kuinka monta tehtävää käyttäjä on suorittanut. Palaute perustuu tietoon, jota ohjelma käyttäjän toiminnasta tallentaa. Ohjelmaa voisi kehittää siten, että suoritetuista tehtävistä tallennetaan enemmän tietoa, jolloin käyttäjälle voisi esittää monipuolisempia tilastotietoja tämän harjoitteluhistoriasta.

Tähän työhön ei kuulunut varsinaisesti oppimistuloksien tehokkuuden mittaaminen. Esimerkiksi parasta harjoittelutapaa voisi yrittää selvittää mittaamalla koehenkilöiden edistymistä eri menetelmien avulla.

Ohjelma laadittiin sillä oletuksella, että sitä käyttää vain yksi käyttäjä. Tämä teki myös toteuttamisesta helpompaa. Opetuskäytössä voi kuitenkin olla tarpeen, että samaa ohjelmaa samalla koneella käyttää useampi henkilö. Ohjelmaa voisikin kehittää siten, että ohjelmaa käytetään erillisten käyttäjätilien kautta, jolloin ohjelma voisi pitää kirjaa eri käyttäjien edistymisestä.

Hyvälaatuisten tehtävien laatiminen harjoituksiin vaatii vaivaa. Yksinkertaisen kaavan noudattaminen ei välttämättä riitä laadukkaan harjoittelumateriaalin tuottamiseen, vaan materiaalin täytyy pohjautua oikeaan musiikkiin. Tässä työssä tehtäviä laadittiin sekä käsin että koneluettavia musiikkiaineistoja tietokoneohjelmalla käsittelemällä. Koneluettava aineistoja on nykyään runsaasti saatavilla Internetin välityksellä, mikä mahdollistaa tehtävien laadinnassa vaadittavan käsin tehtävän työn automatisoinnin.

Lähteet

1 Suuri musiikkitietosanakirja 6. 1992. Keuruu: Weilin+Göös.

2 Virtamo, Keijo toim. 1997. Otavan musiikkitieto, uud.laitos 1.painos.

3 Tumblin, John E. Automatic ear training apparatus. U.S. Patent 4321853. Filing date: 30 Jul 1980, Issue date: 30 Mar 1982.

4 Super Memory: Forget about forgetting. Verkkodokumentti. SuperMemo World. <http://www.supermemo.com>. Luettu 11.03.2013.

5 Elmes, Damien. Anki - friendly, intelligent flashcards. Verkkodokumentti. <http://ankisrs.net/>. Luettu 11.03.2013.

6 Thalheimer, Will. 2006. Spacing Learning Over Time. Verkkodokumentti. <http://www.work-learning.com/catalog.html>. Luettu 15.10.2012.

5 Schiller, Scott. SoundManager 2: Javascript Drum Machine Demo. Verkkodokumentti. <http://www.schillmania.com/projects/soundmanager2/demo/mpc/>. Luettu 19.11.2012.

6 Web Audio Examples. Verkkodokumentti. Google Inc. <http://chromium.googlecode.com/svn/trunk/samples/audio/index.html>. Luettu 19.11.2012.

7 GNU Solfege - free ear training software. Verkkodokumentti. <http://www.solfege.org>. Luettu 9.03.2013.

8 Benbassat, Alain. Free ear training tools for musicians. Verkkodokumentti. <http://www.miles.be>. Luettu 11.03.2013.

9 Telesco, Paula. 1991. Contextual Ear Training. Journal of Music Theory Pedagogy, VOLUME FIVE, No. 2, FALL 1991. s. 179–190.

10 Singer, Frank. The "Charlie Banacos" Exercise. Verkkodokumentti: <http://www.miles.be/articles/14-the-charlie-banacos-exercise>. Luettu 3.3.2013.

11 EarMaster Pro 6 - Ear training and sight-singing software. Verkkodokumentti. EarMaster ApS.. <http://www.earmaster.com>. Luettu 9.03.2013.

12 Auralia. Verkkodokumentti. Avid Technology, Inc. <http://www.sibelius.com/products/auralia>. Luettu 9.03.2013.

13 Issue 8201: Expose Midi Streaming capability of Sonivox Synthesizer. Verkkodokumentti. <http://code.google.com/p/android/issues/detail?id=8201>. Luettu 3.3.2013.

14 Koelle, David. JFugue - Java API for Music Programming. Verkkodokumentti. <http://www.jfugue.org>. Luettu 11.03.2013.

15 Campin, Jack. Verkkodokumentti. <http://www.campin.me.uk>. Luettu 3.3.2013.

16 Walshaw, Chris. About ABC notation. Verkkodokumentti. <http://abcnotation.com/about>. Luettu 3.3.2013.

17 de Clercq, Trevor & Temperley, David. A Corpus Study of Rock Music. Verkkodokumentti. <http://theory.esm.rochester.edu/rock_corpus>. Luettu 3.3.2013.

18 Leitner System for Studying Flashcards. Verkkodokumentti. Mometrix Media LLC. <http://www.flashcardsecrets.com/leitner.htm>. Luettu 11.03.2013.

19 Mizuno, Rika. 2003. The Application of Cognitive Scientific Findings Related to Spacing Effects to Web-Based CAI Systems. Proceedings of the Joint Workshop of Cognition and Learning through Media-Communication for Advanced e-Learning. Berlin: Japanese-German Center Berlin. Saatavilla: <http://psy.isc.chubu.ac.jp/~mizunor/abstract/g3_18_japan_german.pdf>.

20 Inkscape, Draw Freely. Verkkodokumentti. <http://inkscape.org>. Luettu 11.03.2013.

21 Nielsen, Jakob. 1993. Usability Engineering. Academic Press.

22 MigLayout - Java Layout Manager for Swing and SWT. Verkkodokumentti. MiG InfoCom AB. <http://www.miglayout.com>. Luettu 11.03.2013.

23 Introducing JSON. Verkkodokumentti. <http://www.json.org/>. Luettu 11.03.2013.

24 Koskimies, Kai & Mikkonen, Tommi. 2005. Ohjelmistoarkkitehtuurit. Talentum Media Oy.

25 How to Use CardLayout. Verkkodokumentti. Oracle. <http://docs.oracle.com/javase/tutorial/uiswing/layout/card.html>. Luettu 11.03.2013.

26 Jackson JSON Processor. Verkkodokumentti. Codehaus. <http://jackson.codehaus.org>. Luettu 11.03.2013.